البرمجة باستخدام

لغة فري باسكال – بيئة لازاروس

Free Pascal – Lazarus IDE



إعداد

م. أبوبكر شرف الدين سويدان

2015

لازاروس Lazarus

هي بيئة تطويرية متكاملة، متعددة المنصات، حرة ومجانية، مفتوحة المصدر. تستخدم مترجم فري باسكال الذي يدعم أوبجكت باسكال. وإن سألك أحدهم بم تطور برامجك، قل له: أستخدم لغة فري باسكال – بيئة لازاروس.



يمكن لمطوري سطح المكتب استخدام فري باسكال – بيئة لازاروس لتطوير تطبيقات تعمل على عدة أنظمة تشغيل مثل وندوز ولينوكس وماك.

نتاج البرمجة في فري باسكال – بيئة لازاروس هي ملفات تنفيذية طبيعية Native بأحجام صغيرة نسبياً، سريعة التنفيذ، خفيفة على الذاكرة.

تشبه بيئة لازاروس إلى حد كبير بيئة دلفي لتطوير التطبيقات، حيث توفر مجموعة كبيرة من الأدوات والعناصر المساعدة على برمجة التطبيقات بسهولة بالغة.

توفر بيئة لازاروس عناصر تساعد على إنشاء واجهة المستخدم بحيث ما تراه هو ما تحصل عليه، ومحرر للكود، والتنسيق، ومتتبع الأخطاء، وإدارة المشاريع.

كما توفر إمكانية الاتصال والتعامل مع مختلف أنواع قواعد البيانات مثل MySql وPostgresql وSqlite3 وFireBird وOracle وغيرها.

العمل على هذه البيئة سهل وممتع، ولا يحتاج منك إلا الإلمام بأساسيات لغة فري باسكال، التي لها مستندات ومجتمع زاخر بالمعلومات.

من الأمور الممتعة في فري باسكال – بيئة لازاروس، أنه يمكنك برمجة مشروع تطبيق على وندوز مثلاً، وتنتج به برنامجاً يعمل على أي جهاز به نظام وندوز بمجرد نسخ الملف التنفيذي (وبعض الملفات الضرورية للعمل مثل ملف قاعدة البيانات أو ملفات التقارير) وتشغيله دون أية مشاكل تذكر.

كما يمكنك نسخ الكود البرمجي للمشروع ونقله إلى منصة أخرى مثل لينوكس أوبونتو، وعمل Compile فينتج تطبيقاً يعمل على لينوكس دون تغيير حقيقي في الكود! هذا هو معنى الشعار Write once, Compile Anywhere.



لتنزيل نسختك من البيئة، ادخل موقع البيئة : www.lazarus-ide.org

وللمزيد من المعلومات عن لغة فري باسكال، يمكنك مطالعة الموقع الخاص بها: www.freepascal.org

الباب الأول

أساسيات لغة فري باسكال

أولاً - المتغيرات Variables

المتغيرات هي عبارة عن أوعية تحمل مختلف أنواع البيانات. ويجب تعريف المتغيرات في البرنامج قبل استخدامها. ولتسمية متغير، يجب التأكد من أن الاسم يخضع لشروط تسمية المتغير وهي:

- أن يبدأ بحرف إنجليزي.
- يمكن أن يحتوي على أرقام.
- يمكن أن يحتوي على الشرطة التحتية Underscore.
- لا يمكن أن يحتوي على إحدى العلامات الخاصة مثل (~ ! @ # $ % ^ & * ( ) + ` - = { } [ ] : " ; ' < > ? , . / | \ ).

تعريف المتغيرات

يتم تعريف المتغيرات في البرنامج بناءً على مدى صلاحية هذه المتغيرات، فهناك متغيرات على مستوى الإجراء Procedure أو الدالة Function، وهناك متغيرات على مستوى الوحدة Unit الحالية Private Variables، وهناك متغيرات على مستوى المشروع Project وتسمى Public Variables ويمكن التعامل معها من أي مكان ضمن المشروع الحالي.

لتعريف المتغيرات ضمن الإجراءات أو الدوال، قبل كلمة Begin نكتب:

```
Var
        Var1, Var2, Var3 : Data Type;
```

حيث Var1, Var2, Var3 هي أسماء المتغيرات. و Data Type هو نوع البيانات التي ستحملها هذه المتغيرات.

ولتعريف المتغيرات على مستوى الوحدة، نذهب إلى أعلى الكود الخاص بالوحدة، وتحت كلمة Private نعرف المتغيرات ودون استعمال كلمة Var بالصورة التالية:

```
  private
    Var1: Data Type;
    Var2: Data Type;
```

أما لتعريف المتغيرات على مستوى المشروع، فنذهب تحت الكلمة Public ونعرفها بالصورة التالية:

```
  public
    Var1: Data Type;
    Var2: Data Type;
```

أنواع البيانات في فري باسكال – بيئة لازاروس

بشكل أساسي، توجد خمس أنواع للبيانات في فري باسكال – بيئة لازاروس، وهي:

- **Integer**: للقيم الصحيحة.
- **Real**: للقيم الكسرية التي تحتوي على فاصلة عشرية.
- **Boolean**: للقيم المنطقية True وFalse.
- **Char**: للقيم النصية التي تمثل حرفاً واحداً.
- **String**: للقيم التي تمثل سلاسل نصية.

1. الأعداد الصحيحة Integers

تنقسم الأعداد الصحيحة في فري باسكال – بيئة لازاروس إلى أنواع فرعية بناءً على حجم هذه الأعداد بالصورة التالية:

integer types		
Type	**Range**	**Bytes**
Byte	0 .. 255	1
Shortint	-128 .. 127	1
Smallint	-32768 .. 32767	2
Word	0 .. 65535	2
Integer	smallint or longint	2 or 4
Cardinal	longword	4
Longint	-2147483648 .. 2147483647	4
Longword	0..4294967295	4
Int64	-9223372036854775808 .. 9223372036854775807	8
QWord	0 .. 18446744073709551615	8

2. الأعداد الكسرية Real

تنقسم الأعداد الكسرية في فري باسكال – بيئة لازاروس إلى أنواع فرعية بناءً على حجم هذه الأعداد بالصورة التالية:

real types			
Type	**Range**	**Significant digits**	**Bytes**
Real	platform dependent	???	4 or 8
Single	1.5E-45 .. 3.4E38	7-8	4
Double	5.0E-324 .. 1.7E308	15-16	8
Extended	1.9E-4932 .. 1.1E4932	19-20	10
Comp	-2E64+1 .. 2E63-1	19-20	8
Currency	-922337203685477.5808	922337203685477.5807	8

3. القيم المنطقية Boolean

وتنقسم القيم المنطقية أيضاً إلى أنواع فرعية وهي:

boolean types		
Type	**Bytes**	**Ord(True)**
Boolean	1	1
ByteBool	1	Any nonzero value
WordBool	2	Any nonzero value
LongBool	4	Any nonzero value

أمثلة

```
Var
        ItemDescription: String;
        UserID: Integer;
        CostPrice: Double;
        ExitFlag: Boolean;
```

الكلمات المحجوزة Reserved Words

وهي كلمات لا يمكن أن تسمى بها المتغيرات لأنها محجوزة من قبل اللغة، ولها وظائفها الخاصة مثل:

1. Turbo Pascal reserved words

الكلمات المحجوزة من قبل توربو باسكال:

absolute	and	array	asm	begin	break	case	const
constructor	continue	destructor	div	do	downto	else	end
file	for	function	goto	if	implementation	in	inherited
inline	interface	label	mod	nil	not	object	of
on	operator	or	packed	procedure	program	record	reintroduce
repeat	self	set	shl	shr	string	then	to
type	unit	until	uses	var	while	with	xor

2. Delphi reserved words

والكلمات المحجوزة من قبل دلفي إضافة إلى ما سبق هي:

as	class	except	exports	finalization	finally	initialization
is	library	on	property	raise	threadvar	try

3. Free Pascal reserved words

والكلمات المحجوزة من قبل فري باسكال إضافة إلى ما سبق هي:

dispose	exit	false	new	true

وكذلك:

abs	arctan	boolean	char	cos	dispose	eof	eoln
exp	false	input	integer	ln	maxint	new	odd
ord	output	pack	page	pred	read	readln	real
reset	rewrite	round	sin	sqr	sqrt	succ	text
true	trunc write	writeln					

تعيين قيم المتغيرات Assignment

بعد أن تم تعريف المتغير، يمكن تعيين قيمة له تكون من نفس نوعه. الصورة العامة لتعيين قيم المتغيرات هي:

```
Variable_Name := Expression;
```

أمثلة:

```
Var
ItemDescription : String;
TotalPrice, CostPrice, Taxes : Double;
Quantity : Integer;
…
ItemDescription := 'Milk';
TotalPrice := CostPrice * Quantity;
Taxes := TotalPrice * 0.025;
```

المعاملات الرياضية Arithmetic Operators

وهي العمليات التي يمكن إجراؤها على الأرقام سواء كانت صحيحة أو كسرية.

المعامل	وصف العملية	نوع المتغيرات	النتيجة
+	عملية جمع	صحيحة و/أو كسرية	صحيحة و/أو كسرية
-	عملية طرح	صحيحة و/أو كسرية	صحيحة و/أو كسرية
*	عملية ضرب	صحيحة و/أو كسرية	صحيحة و/أو كسرية
/	عملية قسمة	صحيحة و/أو كسرية	كسرية
Div	عملية قسمة بناتج صحيح	صحيحة	صحيحة
Mod	المتبقي من القسمة	صحيحة	صحيحة

ثانياً – الثوابت Constants

الثوابت على معرفات بقيم ثابتة، لا تتغير في البرنامج. وعند تسميتها وتعريفها، ينطبق عليها كل ما ينطبق على تسمية المتغيرات.

تعريف الثوابت في البرنامج

يتم تعريف الثوابت بالصورة التالية:

```
const
  Identifier1 : Data Type;
  Identifier2 : Data Type;
  Identifier3 : Data Type;
```

ويمكن تعيين القيم لهذه الثوابت من خلال التعريف نفسه بالصورة التالية:

```
const
  Identifier1 : Data Type = Value;
  Identifier2 : Data Type = Value;
  Identifier3 : Data Type = Value;
```

أمثلة:

```
Const
Pi : Real = 3.14;
Store_Name : String = 'Main Store';
```

وكما نلاحظ، فعند تعيين القيم أثناء التعريف لا نستخدم (=:) بل نستخدم (=) فقط.

ثالثاً– التعابير المنطقية Boolean Expressions

تستخدم التعابير المنطقية عند مقارنة قيمتين أو أكثر، بحيث يكون الناتج هو قيمة منطقية True أو False. ومعاملات التعابير المنطقية الأساسية هي:

المعامل	المعنى
<	يعطي True إذا كانت القيمة الأولى أصغر من القيمة الثانية وإلا سيعطي False
>	يعطي True إذا كانت القيمة الأولى أكبر من القيمة الثانية وإلا سيعطي False
=	يعطي True إذا كانت القيمة الأولى تساوي القيمة الثانية وإلا سيعطي False
<=	يعطي True إذا كانت القيمة الأولى أصغر من أو تساوي القيمة الثانية وإلا سيعطي False
>=	يعطي True إذا كانت القيمة الأولى أكبر من أو تساوي القيمة الثانية وإلا سيعطي False
<>	يعطي True إذا كانت القيمة الأولى لا تساوي القيمة الثانية وإلا سيعطي False

وهناك معاملات أكثر تعقيداً من المعاملات الأساسية وهي:

المعامل	المعنى
Not	عكس النتيجة المعطاة
And	تعطي True في حال كانت النتيجة True ، وإلا ستعطي False.
Or	تعطي True إذا كانت إحدى النتائج True، وإلا ستعطي False
Xor	تعطي True إذا اختلفت النتائج، وإلا ستعطي False

```
NOT:
Not (True) = False
Not (False) = True
```

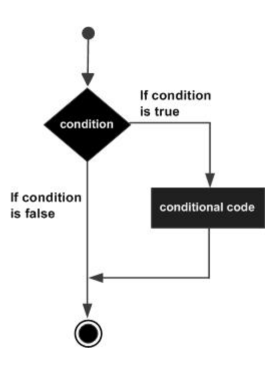
```
AND:
True AND True = True
True AND False = False
```

```
OR:
True OR True = True
True OR False = True
False OR True = True
False OR False = False
```

```
XOR:
True XOR True = False
True XOR False = True
False XOR True = True
False XOR False = False
```

رابعاً – اتخاذ القرارات Decision Making

1. الجملة If - Then statement



نستخدم جملة If-Then عندما نريد تنفيذ كود ما نتيجة لتحقق شرط معين.

الشكل العام لجملة if-Then:

```
If (condition) then
        Statement;
```

وبذلك، سيتم تنفيذ الـ Statement فقط إذا تحقق الـ Condition، وإلا سيتم تجاهل الـ Statement وتنفيذ الكود الذي يليها.

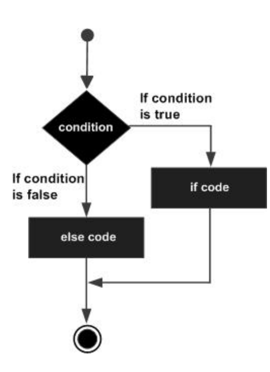
وفي حالة كان الكود الذي نريد تنفيذه عند تحقق الشرط طويلاً (أكثر من جملة واحدة)، نستخدم الصيغة التالية:

```
If (Condition) then
Begin
   Statement1;
   Statement2;
   Statement3;
…
   Statement(n);
End;
```

2. الجملة If – Then - Else statement

هي نفسها جملة If-Then مضافاً إليها خيار Else.

نستخدم جملة If-Then-Else عندما نريد تنفيذ كود ما نتيجة لعدم تحقق الشرط.



الشكل العام لجملة if-Then-Else:

```
If (condition) then
        Statement1
Else
        Statement2;
```

وبذلك، سيتم تنفيذ الـ Statement2 فقط إذا لم يتحقق الـ Condition.

لاحظ أن الـ Statement1 التي تسبق (Else) لا تتبعها فاصلة منقوطة (;) وهي قاعدة يجب الانتباه إليها دائماً: (**أي جملة تسبق Else لا تُتبع بفاصلة منقوطة**).

وبالطبع، إن كان الكود المطلوب تنفيذه في حالة عدم تحقق الشرط طويلاً (أكثر من جملة واحدة)، نستخدم الصورة التالية:

```
If (condition) then
        Statement1
Else
        Begin
                Statement1;
                Statement2;
                Statement(n);
        End;
```

3. جملة Nested If Statement

من المتاح تشابك جمل If في البرنامج في لغة فري باسكال، والذي يعني استعمال جمل If-Else داخل جمل If-Else أخرى.

نستخدم جمل If المتداخلة عدما نريد تنفيذ كود ما اعتماداً على صحة أو خطأ شرط معين.

الشكل العام لجملة if المتداخلة:

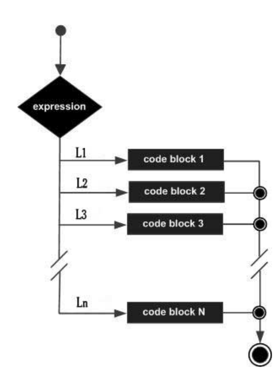
```
If (condition1) then
        Begin
           If (Condition2) then Statement1;
        End
Else
        Statement2;
```

وبذلك، سيتم تنفيذ الـ Statement1 فقط إذا تحقق الشرط الأول Condition1 (و) تحقق الشرط الثاني Condition2.

ويمكن أن يكون التشبيك في الجزء Else بالصورة التالية:

```
If (condition1) then
        Begin
           If (Condition2) then Statement1;
        End
Else
        Begin
            If (Condition3) then
                Begin
                        Statement2;
                End;
        End;
```

4. جملة Case Statement



وهي صورة أخرى للجملة if، ونستخدمها عند التحقق من قيم معينة معروفة مسبقاً.

الشكل العام لجملة Case Statement:

```
case (expression) of
   L1 : S1;
   L2: S2;
   ...
   ...
   Ln: Sn;
end;
```

حيث : L1, L2 هي القيمة الناتجة عن الشرط. وS1, S2 هي الجمل التي يتم تنفيذها في كل حالة.

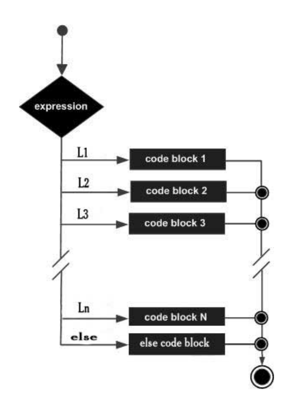
مثال:

```
program checkCase;
var
   grade: char;
begin
   grade := 'A';

   case (grade) of
      'A' : writeln('Excellent!' );
      'B', 'C': writeln('Well done' );
      'D' : writeln('You passed' );
      'F' : writeln('Better try again' );
   end;

   writeln('Your grade is  ', grade );
end.
```

5. جملة Case-Else Statement



ونستخدمها في حالة كون نتيجة الشرط مختلفة عن القيم المتوقعة (في حالة صحة الشرط).

الشكل العام لجملة Case-Else Statement:

```
case (expression) of
   L1 : S1;
   L2 : S2;
   ...
   ...
   Ln: Sn;
else
   Sm;
end;
```

حيث : L1, L2 هي القيمة الناتجة عن الشرط. وS1, S2 هي الجمل التي يتم تنفيذها في كل حالة. أما Sm فهي الجملة التي تنفذ في ما عدا ذلك.

مثال:

```
program checkCase;
var
   grade: char;

begin
   grade := 'F';
   case (grade) of
      'A' : writeln('Excellent!' );
      'B', 'C': writeln('Well done' );
      'D' : writeln('You passed' );

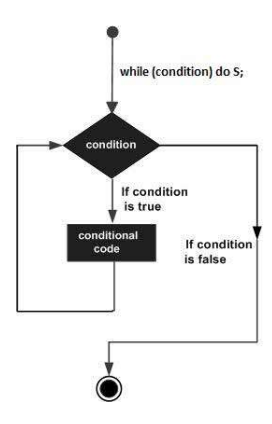
   else
      writeln('You really did not study right!' );
    end;

   writeln('Your grade is  ', grade );
end.
```

خامساً – جمل التكرار Loop

في بعض الأحيان نحتاج إلى تنفيذ عدد من الجمل عدة مرات حتى تحقق شرط معين. وعند تحقق الشرط اللازم للتوقف، نستأنف تنفيذ الجمل والعبارات التي تليها. توفر لغة فري باسكال العديد من جمل التكرار، أهمها:

1. حلقة التكرار While-do Loop



ونستخدمها لتنفيذ جملة أو مجموعة من الجمل **طالما** تحقق الشرط.

الشكل العام:

```
while (condition) do Statement1;
```

وطالما كان الشرط متحققاً (أي True) . ولا بد من توفر جملة تقطع التكرار، وإلا فإنه سيستمر في التكرار إلى مالا نهاية.

مثال:

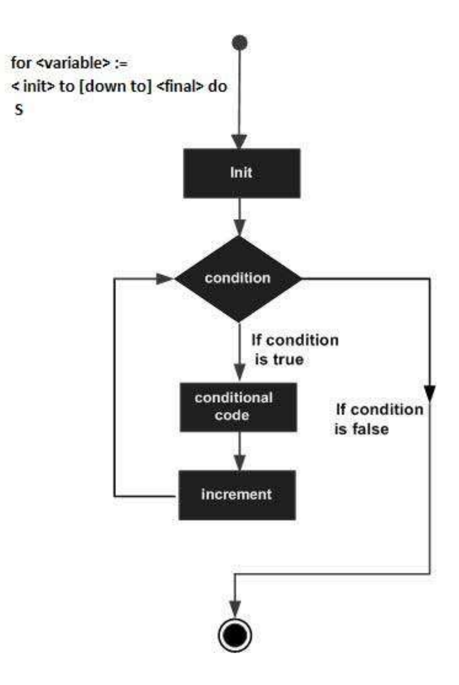
```
program whileLoop;
var
   a: integer;

begin
   a := 10;
   while  a < 20  do

   begin
      writeln('value of a: ', a);
      a := a + 1;
   end;
end.
```

لاحظ الجملة (a := a + 1;)، فهي المسؤولة عن التسبب في توقف التكرار.

2. حلقة التكرار For-Do Loop



ونستخدمها لتكرار نفيذ جملة أو مجموعة من الجمل في مدى معين **تصاعدياً أو تنازلياً**.

الشكل العام:

```
for <variable-name> := <initial_value> to [down to] <final_value> do
   S;
```

ومن خلال الشكل العام، سيكون هناك صيغتين:

```
for <variable-name> := <initial_value> to <final_value> do
   S;
```

و

```
for <variable-name> := <initial_value> down to <final_value> do
   S;
```

مثال:

```
program forLoop;
var
   a: integer;

begin
   for a := 10  to 20 do

   begin
      writeln('value of a: ', a);
   end;
end.
```

النتيجة ستكون:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```

أو بالصورة الثانية

```
program forLoop;
var
   a: integer;

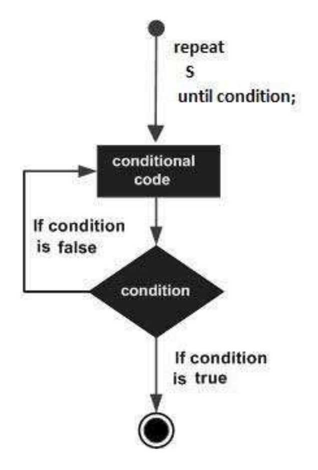
begin
   for a := 20  down to 10 do

   begin
      writeln('value of a: ', a);
   end;
end.
```

والنتيجة ستكون:

```
value of a: 20
value of a: 19
value of a: 18
value of a: 17
value of a: 16
value of a: 15
value of a: 14
value of a: 13
value of a: 12
value of a: 11
value of a: 10
```

3. حلقة التكرار Repeat-Until Loop



ونستخدمها لتكرار تنفيذ جملة أو مجموعة من الجمل **حتى يتحقق شرط التوقف**.

الشكل العام:

```
repeat
   S1;
   S2;
   ...
   ...
   Sn;
until condition;
```

مثال:

```
program repeatUntilLoop;
var
   a: integer;

begin
   a := 10;
   (* repeat until loop execution *)
   repeat
      writeln('value of a: ', a);
      a := a + 1
   until a = 15;
end.
```

والنتيجة هي:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
```